

DATENBANKANWENDUNG

Wintersemester 2013/2014

Holger Schwarz
Universität Stuttgart, IPVS
holger.schwarz@ipvs.uni-stuttgart.de

Beginn: 23.10.2013
Mittwochs: 11.45 – 15.15 Uhr, Raum 46-268 (Pause 13.00 – 13.30)
Donnerstags: 10.00 – 11.30 Uhr, Raum 46-268
11.45 – 13.15 Uhr, Raum 46-260

<http://www.lgis.informatik.uni-kl.de/cms/courses/datenbankanwendung/>



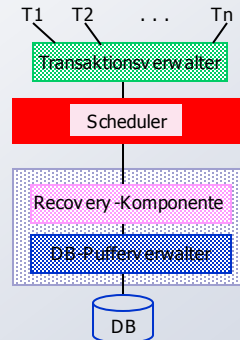
8. Sperrverfahren – Implementierung und Leistungsanalyse¹

- **Synchronisation – Überblick über die Verfahren**
- **Zweiphasen-Sperrprotokolle**
 - RUX-Protokoll
 - Hierarchische Sperrverfahren
- **Einführung von Konsistenzebenen**
 - Reduzierung von TA-Blockierungen, aber Programmierdisziplin gefordert
 - Unterschiedliche Ansätze basierend auf
 - Sperrdauer für R und X
 - zu tolerierende "Fehler"
- **Optimierungen**
 - RA X und RAC
 - Mehrversionen-Verfahren
 - Prädikatssperren
 - Synchronisation auf Objekten
 - Spezielle Synchronisationsprotokolle
- **Leistungsanalyse und Bewertung**
 - Ergebnisse empirischer Untersuchungen
 - Dynamische Lastkontrolle

1. Thomasian, A.: Concurrency Control: Methods, Performance and Analysis, in: ACM Computing Surveys 30:1, 1998, 70-119, 8-2

Einbettung des DB-Schedulers

■ Stark vereinfachtes Modell



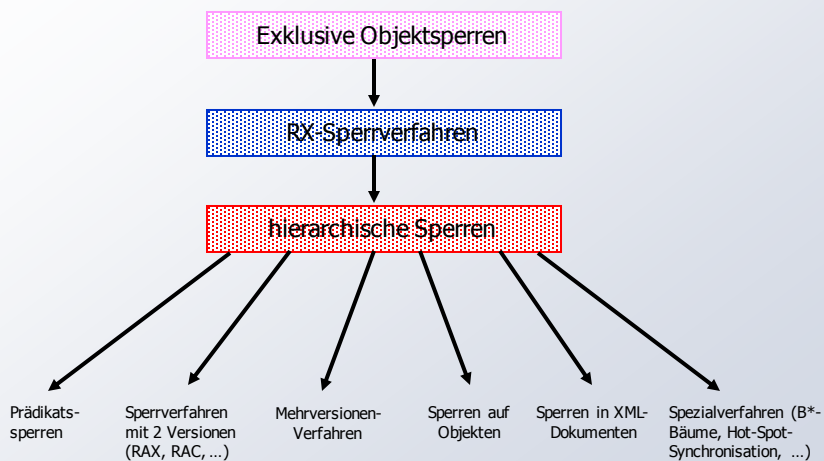
■ Welche Aufgaben hat der Scheduler (Sperrverwalter, Lock-Mgr)?

- als Komponente der Transaktionsverwaltung zuständig für I von ACID
- kontrolliert die beim TA-Ablauf auftretenden Konfliktooperationen (R/W, W/R, W/W) und garantiert insbesondere, dass nur „serialisierbare“ TAs erfolgreich beendet werden
- verhindert nicht-serialisierbare TAs
- kann jedoch keine Deadlocks verhindern. Zur Deadlock-Auflösung ist eine Kooperation mit der Recovery-Komponente erforderlich (Rücksetzen von TAs)

⇒ Garantiert „vernünftige“ Schedules

8-3

Historische Entwicklung von Synchronisationsverfahren



8-4

Phantom-Problem

- **Einfügungen oder Löschungen können zu falschen Schlussfolgerungen verleiten**

Lesetransaktion (Gehaltssumme überprüfen)	Änderungstransaktion (Einfügen eines neuen Angestellten)
<pre>SELECT SUM (Gehalt) INTO :summe FROM Pers WHERE Anr = 17 SELECT Gehaltssumme INTO :gsumme FROM Abt WHERE Anr = 17 IF gsumme <> summe THEN <Fehlerbehandlung></pre>	<pre>INSERT INTO Pers (Pnr, Anr, Gehalt) VALUES (4567, 17, 55.000) UPDATE Abt SET Gehaltssumme = Gehaltssumme + 55.000 WHERE Anr = 17</pre>
	Zeit ↓

8-5

Sperrverfahren

- **Sperrenbasierte Synchronisation**
 - Sperren stellen während des laufenden Betriebs sicher, dass die resultierende Historie serialisierbar bleibt
 - Es gibt mehrere Varianten
 - **Zur Realisierung der Synchronisation gibt es viele verschiedenartige Sperrverfahren**
 - **Sperrenbasierte Verfahren auf physischen DB-Objekten**
Bei einer Konfliktoperation blockieren sie den Zugriff auf das Objekt.
Verfeinerungen: Konversion des Spermodus, multiple Sperrgranulate
 - **Versionsverfahren**
Es werden zwei oder mehr Versionen eines Objektes gehalten.
Keine Behinderung der Leser durch Schreiber
 - **Prädikatssperren**
Es wird die Menge der möglichen Objekte, die das Prädikat erfüllen, gesperrt.
Elegante Lösung für das Phantom-Problem
 - **Sperren auf Objekten**
Nutzung der Kommutativität von Änderungen
 - **Spezielle Synchronisationsverfahren**
Nutzung der Semantik von Änderungen
 - ...
- ⇒ **Sperrverfahren sind pessimistisch und universell einsetzbar**

8-6

RX-Sperrverfahren

Sperrmodi

- Sperrmodus des Objektes: NL (no lock), R (read), X (exclusive)
- Sperranforderung einer Transaktion: R, X

Kompatibilitätsmatrix

		aktueller Modus des Objekts			Modus-Übergang				
			NL	R	X		NL	R	X
angeforderter Modus der TA	R	+	+	-	R	R	R	-	
	X	+	-	-	X	X	-	-	

- Falls Sperre nicht gewährt werden kann, muss die anfordernde TA warten, bis das Objekt freigegeben wird (Commit/A bort der die Sperre besitzenden TA)
- Wartbeziehungen werden in einem *Wait-for-Graph* (WFG) verwaltet

Erinnerung

- Zweiphasigkeit: 2PL
- Unter striktem 2PL (S2PL) werden alle exklusiven Sperren einer TA bis zu ihrer Terminierung gehalten
- Unter starkem 2PL (SS2PL) werden alle Sperren einer TA bis zu ihrer Terminierung gehalten

Scheduling von Sperranforderungen

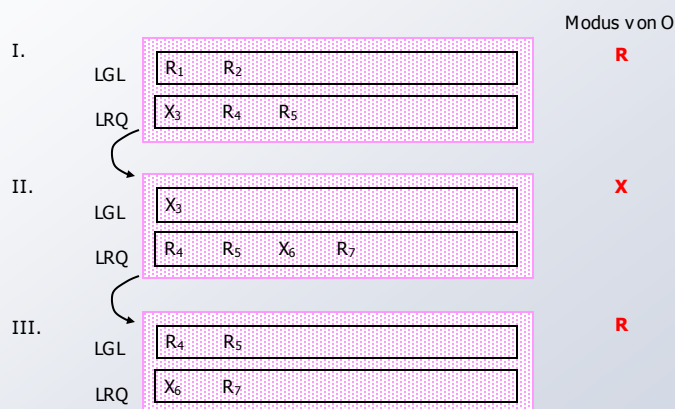
- pro Objekt existiert eine Liste der gewährten Sperranforderungen **LGL** (list of granted locks) und eine Warteschlange **LRQ** (lock request queue)
- Wann wird eine Sperranforderung LR (lock request) gewährt?
- Legende: Der Index *i* gibt die zeitliche Reihenfolge der Sperranforderung und gleichzeitig eine Transaktionsnummer an. Alle Sperranforderungen beziehen sich auf ein Objekt *O*.

8-7

RX-Sperrverfahren (2)

Faires Scheduling (First Come, First Serve, FCFS)

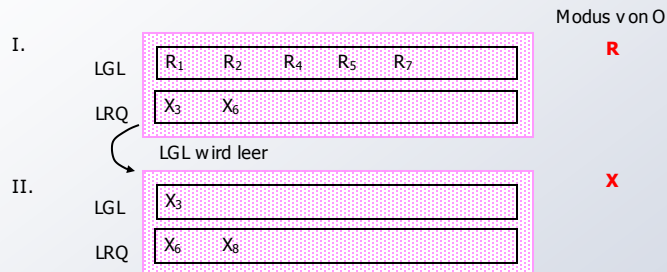
- LR wird gewährt
 - wenn LRQ leer und LR kompatibel mit dem aktuellen Modus von *O*,
 - wenn LGL leer, erster LR aus LRQ



RX-Sperrverfahren (3)

Unfares Scheduling

- LR wird gewährt, wenn LR kompatibel mit dem aktuellen Modus von O



8-9

2PL

SQL-API

- An der SQL-Schnittstelle ist die Sperranforderung und -freigabe **nicht sichtbar!**
- Objektanforderung durch Zugriffsmodul (nach Optimierung) über OID (direkt), Scan (Fetch Next), Index usw.
- Optimierung der Sperranforderungen im DBMS
 - Mehrere Sperrgranulate anwendbar
 - Nach welchen Kriterien entscheidet das DBMS?
 - Zusätzliche Option: **Sperreskalation**
- Wie wird das Phantom-Problem gelöst?

Anwendung des 2PL-Protokolls

T1	T2	Bem.
BOT lock (a,X) read (a) write (a)		
	BOT lock (a,X)	T2 wartet: WfG
lock (b,X) read (b) unlock (a)		
	read (a) write (a) unlock (a) commit	T2 wecken
unlock (b)		

⇒ Praktischer Einsatz erfordert mindestens striktes 2PL-Protokoll

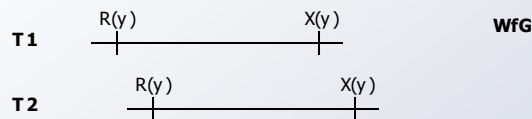
8-10

RUX-Sperrverfahren

Forderung

- Wahl des gemäß der Operation schwächst möglichen Spermodus
- Möglichkeit der Sperrkonversion (upgrade), falls stärkerer Spermodus erforderlich
- Anwendung: viele Objekte sind zu lesen, aber nur wenige zu aktualisieren

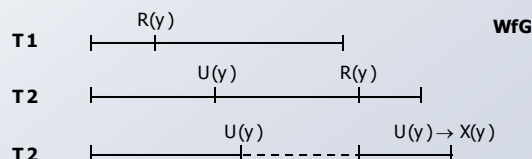
Problem: Sperrkonversionen



Erweitertes Sperrverfahren:

- Ziel: **Verhinderung v on Konversions-Deadlocks**
- U-Sperre für Lesen mit Änderungsabsicht (Prüfmodus)
- bei Änderung Konversion $U \rightarrow X$, andernfalls $U \rightarrow R$ (downgrade)

Wirkungsweise



8-11

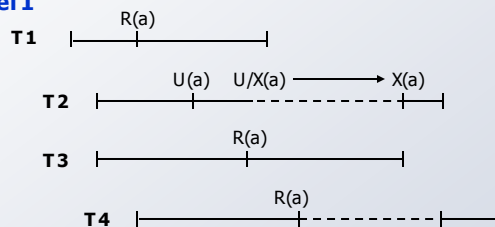
RUX-Sperrverfahren (2)

Symmetrische Variante

- Was bewirkt eine Symmetrie bei U?

	R	U	X
R	+	+	-
U	+	-	-
X	-	-	-

Beispiel 1



Beobachtung

- Bei Downgrade einer U-Sperre (U/R) gibt es keine Probleme, da R mit dem Modus paralleler Leser (R) verträglich ist
- Bei Upgrade einer U-Sperre (U/X) muss die Transaktion warten, bis alle momentanen Leser ihre Sperren auf dem Objekt freigegeben haben
- Bei Warten muss die konvertierende Transaktion an die Spitze der LRQ gesetzt werden. Warum?
- Nach Anmeldung einer Konversion gibt es noch verschiedene Scheduling-Optionen⁸⁻¹²

RUX-Sperrverfahren (3)

Beispiel 2 – verschiedene Scheduling-Strategien

Fair

I.

LGL

R₁ U₂ R₃

LRQ

U₄ R₅

Modus von O

U

Konversionsanforderung U/X nach Eintreffen von R₅:
X₂ wartet auf Konversion

II.

LGL

R₁ R₃

LRQ

U/X₂ U₄ R₅

U

III.

LGL

X₂

LRQ

U₄ R₅ R₆ U₇ R₈

X

IV.

LGL

U₄ R₅ R₆

LRQ

U₇ R₈

U

8-13

RUX-Sperrverfahren (4)

Beispiel 2 – verschiedene Scheduling-Strategien (Cont.)

Pushing (aufdringlich)

I.

LGL

R₁ U₂ R₃ R₅

LRQ

U₄

Modus von O

U

Konversionsanforderung U/X nach Eintreffen von R₅:
X₂ wartet auf Konversion

II.

LGL

R₁ R₃ R₅

LRQ

U/X₂ U₄ R₆

U

III.

LGL

X₂

LRQ

U₄ R₆ U₇ R₈

X

IV.

LGL

U₄ R₆ R₈

LRQ

U₇

U

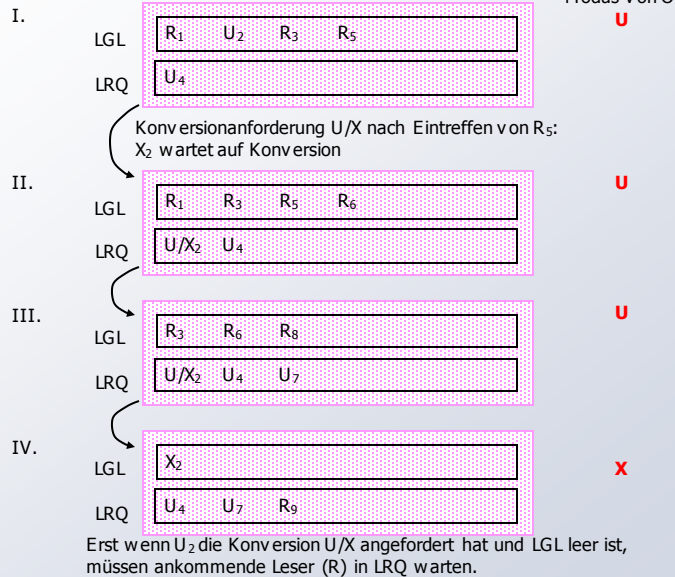
Erst wenn U₂ oder U₄ die Konversion U/X anfordern, müssen ankommende Leser (R) in LRQ warten.

8-14

RUX-Sperrverfahren (5)

Beispiel 2 – verschiedene Scheduling-Strategien (Cont.)

Unfair

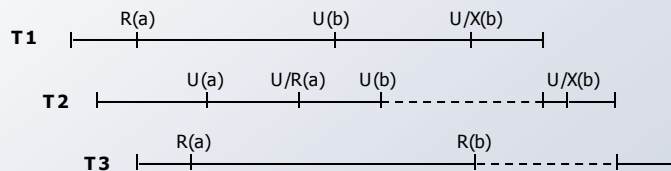


RUX-Sperrverfahren (6)

Unsymmetrie bei U (u.a. in DB2 eingesetzt)

	R	U	X
R	+	-	-
U	+	-	-
X	-	-	-

Beispiel 3

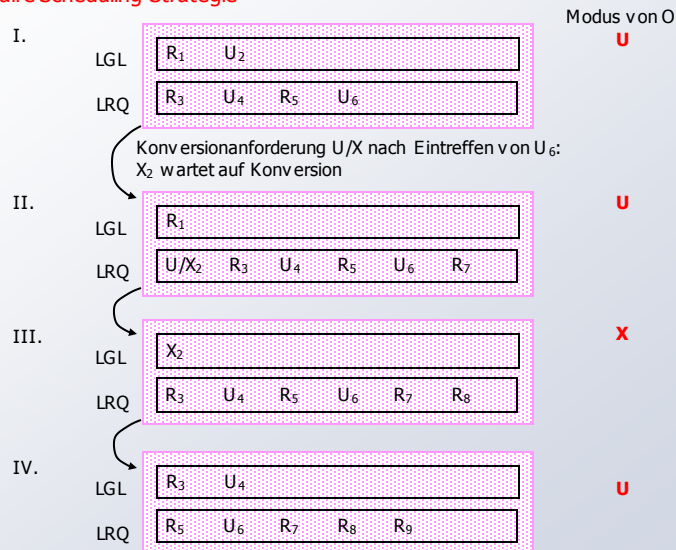


Beobachtung

- Bei Downgrade einer U-Sperre (U/R) gibt es keine Probleme, da R mit dem Modus paralleler Leser (R) verträglich ist
- Bei Upgrade einer U-Sperre (U/X) muss die Transaktion warten, bis alle momentanen Leser ihre Sperren auf dem Objekt freigegeben haben
- RUX-Verfahren **reduziert Konversions-Deadlocks**, ist aber nicht frei von ihnen. R₃(a) U₂(a) U₂(b) U/X₂(a) R₃(b) hätte in Beispiel 3 zu einem Deadlock geführt

RUX-Sperrverfahren (7)

Beispiel 4 Faire Scheduling-Strategie



8-17

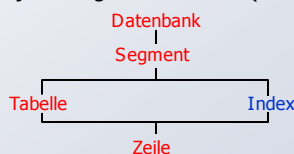
Hierarchische Sperrverfahren

- Sperrgranulat bestimmt Parallelität/Aufwand**
 - Feines Granulat reduziert Sperrkonflikte
 - Jedoch sind viele Sperren anzufordern und zu verwalten
- Hierarchische Verfahren erlauben Flexibilität bei Wahl des Granulates (multi-granularity locking), z. B. Synchronisation**
 - langer TAs auf Tabellenebene
 - kurzer TAs auf Zeilenebene

- Kommerzielle DBS unterstützen zumeist mindestens zweistufige Objekthierarchie, z. B.**



- Verfahren nicht auf reine Hierarchien beschränkt**, sondern auch auf halbgeordnete Objektmengen erweiterbar (siehe auch objektorientierte DBS)



- Verfahren erheblich komplexer als einfache Sperrverfahren**
(mehr Sperrenmodi, Konversionen, Deadlock-Behandlung, ...)

8-18

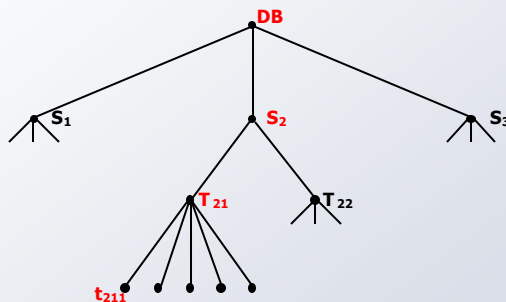
Beispiel einer Sperrhierarchie

Datenbank

Dateien (Segmente)

Satztypen (Tabellen)

Sätze (Zeilen)



Wieviel Aufwand zum Sperren von

- 1 Satz
- k Sätzen
- 1 Satztyp

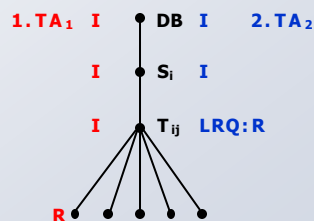
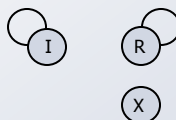
Was ist Sperr-Eskalation?

8-19

Anwartschaftssperren

- **Mit R- und X-Sperre werden alle Nachfolgerknoten implizit mit gesperrt**
⇒ Einsparungen möglich
- **Alle Vorgängerknoten sind ebenfalls zu sperren**, um Unverträglichkeiten zu vermeiden
⇒ Verwendung von Anwartschaftssperren ('intention locks')
- **Allgemeine Anwartschaftssperre (I-Sperre)**

	I	R	X
I	+	-	-
R	-	+	-
X	-	-	-



- **Unverträglich von I- und R-Sperren**

- I-Sperre bei parallelem Lesen zu restriktiv!
- TA₂ wartet (LRQ: Lock Request Queue)

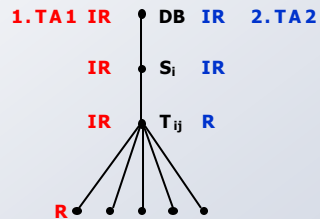
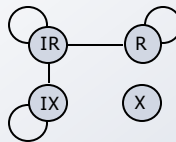
⇒ Zwei Arten von Anwartschaftssperren (IR und IX)

8-20

Anwertschaftssperren (2)

Anwertschaftssperren für Leser und Schreiber

	IR	IX	R	X
IR	+	+	+	-
IX	+	+	-	-
R	+	-	+	-
X	-	-	-	-



- IR-Sperre (*intent read*), falls auf untergeordneten Objekten nur lesend zugegriffen wird, sonst IX-Sperre
- Weitere Verfeinerung sinnvoll, um den Fall zu unterstützen, wo alle Sätze eines Satztyps gelesen und nur einige davon geändert werden sollen
 - X-Sperre auf Satztyp sehr restriktiv
 - IX-Sperre auf Satztyp verlangt Sperren jedes Satzes
- ⇒ neuer Typ von Anwertschaftssperre: **RIX = R + IX**
 - sperrt das Objekt in R-Modus und verlangt
 - X-Sperren auf tieferer Hierarchieebene nur für zu ändernde Objekte

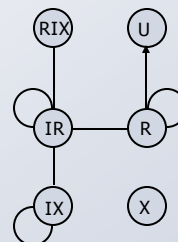
8-21

Anwertschaftssperren (3)

Vollständiges Protokoll der Anwertschaftssperren

- RIX gibt ein Leserecht auf den Knoten und seine Nachfolger. Weiterhin ist damit das Recht verbunden, auf Nachfolger-Knoten IX, U und X-Sperren anzufordern.
- U gewährt ein Leserecht auf den Knoten und seine Nachfolger. Dieser Modus repräsentiert die Absicht, den Knoten in der Zukunft zu verändern. Bei Änderung Konversion $U \rightarrow X$, sonst $U \rightarrow R$.

+	IR	IX	R	RIX	U	X
IR	+	+	+	+	-	-
IX	+	+	-	-	-	-
R	+	-	+	-	-	-
RIX	+	-	-	-	-	-
U	-	-	+	-	-	-
X	-	-	-	-	-	-



Sperrdisziplin erforderlich

- Sperranforderungen v von der Wurzel zu den Blättern
- Bevor TA eine R- oder IR-Sperre für einen Knoten anfordert, muss sie für alle Vorgängerknoten IX- oder IR-Sperren besitzen
- Bei einer X-, U-, RIX- oder IX-Anforderung müssen alle Vorgängerknoten in RIX oder IX gehalten werden
- Sperrfreigaben v von den Blättern zu der Wurzel
- Bei EOT sind alle Sperren freizugeben

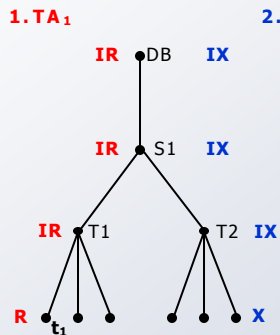
8-22

Hierarchische Sperrverfahren: Beispiele

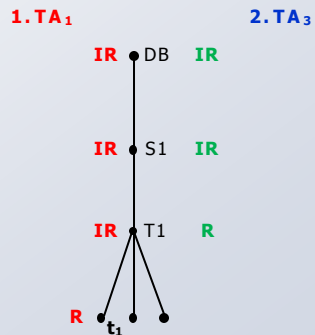
IR- und IX-Modus

- TA₁ liest t₁ in T1
- TA₂ ändert Zeile in T2 (a)
- TA₃ liest T1 (b)

(a)



(b)



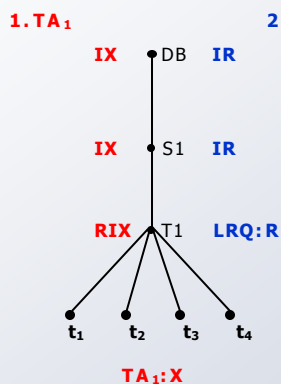
8-23

Hierarchische Sperrverfahren: Beispiele (2)

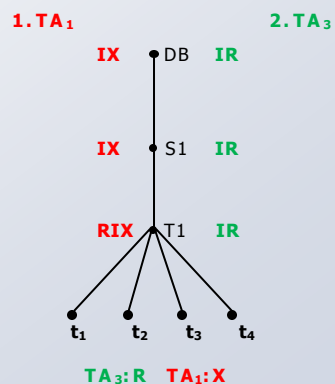
RIX-Modus

- TA₁ liest alle Zeilen von T1 und ändert t₃
- TA₂ liest T1 (a)
- TA₃ liest t₂ in T1 (b)

(a)



(b)



8-24

Sperrverfahren in Datenbanksystemen

- **Aufgabe von Sperrverfahren:** Vermeidung von Anomalien, indem man
 - zu ändernde Objekte dem Zugriff aller anderen Transaktionen entzieht
 - zu lesende Objekte vor Änderungen schützt
- **Standardverfahren:** Hierarchisches Zweiphasen-Sperrprotokoll
 - mehrere Sperrgranulate
 - Verringerung der Anzahl der Sperranforderungen
- **Häufig beobachtete Probleme bei Sperren**
 - Zweiphasigkeit führt häufig zu langen Wartezeiten (starke Serialisierung)
 - blinde Durchsatzmaximierung?

Um Durchsatzziel zu erreichen:
mehr aktive TAs → **mehr** gesperrte Objekte → **höhere** Konflikt-WS →
längere Sperrwartezeiten, **höhere** Deadlock-Raten → **noch mehr** aktive TAs

 - Häufig berührte Objekte können zu Engpässen (*Hot Spots*) werden
 - Eigenschaften des Schemas können *High-Traffic-Objekte* erzeugen
- **Einführung von Konsistenzebenen** zur Reduktion des Blockierungspotentials:
 - ⇒ **Programmierdisziplin gefordert!**
- **Optimierungen!?**
 - Änderungen auf privaten Objektkopien (verkürzte Dauer exklusiver Sperren)
 - Nutzung mehrerer Objektversionen
 - Prädikatssperren, Präzisionssperren
 - spezialisierte Sperren

8-25

Konsistenzebenen

- **Serialisierbare Abläufe**
 - gewährleisten „automatisch“ Korrektheit des Mehrbenutzerbetriebs
 - erzwingen u. U. lange Blockierungszeiten paralleler Transaktionen
 - „Schwächere“ Konsistenzebene bei der Synchronisation von Leseoperationen erlaubt höhere Parallelitätsgrade und Reduktion von Blockierungen, erfordert aber Programmierdisziplin!

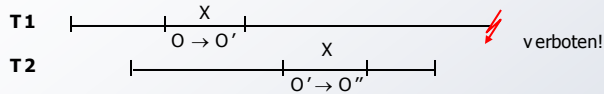
⇒ **Inkaufnahme von Anomalien reduziert die TA-Behinderungen**
- **Konsistenzebenen** (basierend auf verkürzten Sperrdauern)
 - Ebene 3:** Transaktion T sieht Konsistenzebene 3, wenn gilt:
 - a) T verändert keine schmutzigen Daten anderer Transaktionen
 - b) T gibt keine Änderungen vor EOT frei
 - c) T liest keine schmutzigen Daten anderer Transaktionen
 - d) Von T gelesene Daten werden durch andere Transaktionen erst nach EOT von T verändert
 - Ebene 2:** Transaktion T sieht Konsistenzebene 2, wenn sie die Bedingungen a, b und c erfüllt
 - Ebene 1:** Transaktion T sieht Konsistenzebene 1, wenn sie die Bedingungen a und b erfüllt
 - Ebene 0:** Transaktion T sieht Konsistenzebene 0, wenn sie nur Bedingung a erfüllt

8-26

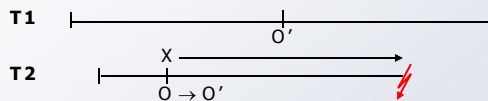
Konsistenzebenen (2)

■ RX-Sperrverfahren und Konsistenzebenen (Beispiele für nur ein Objekt O)

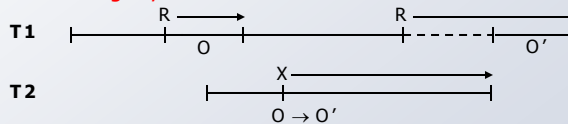
KE 0: kurze X, keine R



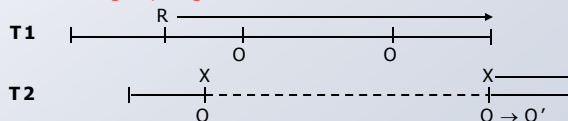
KE 1: lange X, keine R



KE 2: lange X, kurze R



KE 3: lange X, lange R



8-27

Konsistenzebenen (3)

■ **Konsistenzebene 3** (eigentlich KE 2,99)

- wünschenswert, jedoch oft viele Sperrkonflikte wegen langer Schreib- und Lesesperren

■ **Konsistenzebene 2**

- nur lange Schreibesperren, jedoch kurze Lesesperren
- 'unrepeatable read' möglich

■ **Konsistenzebene 1**

- lange Schreibesperren, keine Lesesperren
- 'dirty read' (und 'lost update') möglich

■ **Konsistenzebene 0**

- kurze Schreibesperren ('Chaos')

⇒ Kommerzielle DBS empfehlen meist Konsistenzebene 2

■ **Wahlangebot**

- Einige DBS (Oracle, DB2, MS SQL Server, ...) bieten Wahlmöglichkeit zwischen
 - 'repeatable read' (KE 3) und
 - 'cursor stability' (KE 2)
- Einige DBS bieten auch *BROWSE-Funktion*, d. h. Lesen ohne Setzen von Sperren (KE 1)

8-28

Konsistenzebenen (4)

- SQL erlaubt Wahl zwischen vier Konsistenzebenen (Isolation Level)
- Konsistenzebenen sind durch die Anomalien bestimmt, die jeweils in Kauf genommen werden
 - A abgeschwächte Konsistenzanforderungen betreffen nur Leseoperationen!
 - Lost Update muss generell vermieden werden, d. h., W/W-Abhängigkeiten müssen stets beachtet werden

Konsistenz-ebene	Anomalie		
	Dirty Read	Non-Repeatable Read	Phantome
Read Uncommitted	+	+	+
Read Committed	-	+	+
Repeatable Read	-	-	+
Serializable	-	-	-

- Default ist Serialisierbarkeit² (serializable)

2. Repeatable Read entspricht Read Stability und Serializable entspricht Repeatable Read in DB2

8-29

Konsistenzebenen (5)

- SQL-Anweisung zum Setzen der Konsistenzebene

```
SET TRANSACTION [mode] [ISOLATION LEVEL level]
```

- Transaktionsmodus: READ WRITE (Default) bzw. READ ONLY
- Beispiel**
SET TRANSACTION READ ONLY, ISOLATION LEVEL READ COMMITTED
- READ UNCOMMITTED für Änderungstransaktionen unzulässig

- Was ist der Unterschied zwischen KE3 und "Serializable"?

- Repeatable Read** - Sperren von vorhandenen Objekten

Datenstruktur: O1, O2, ... Zugriff mit Get Next

1. Lesefolge von T1:
Einfügen von T2:
2. Lesefolge von T1:

- Serializable** - garantiert Abwesenheit von Phantomen

1. Lesefolge von T1:
Einfügen von T2:
2. Lesefolge von T1:

8-30

Sperrverfahren mit Versionen (RAX)

Kompatibilitätsmatrix

	R	A	X
R	+	⊕	-
A	⊕	-	-
X	-	-	-

A = Analyse

Ablaufbeispiel

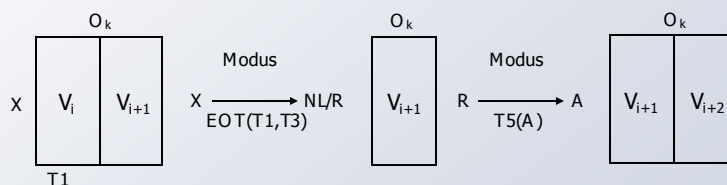
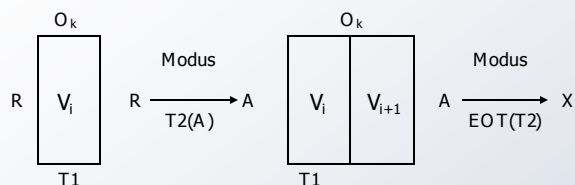


RAX: T2 → T1

8-31

RAX (2)

Änderungen erfolgen in temporärer Objektkopie



LRQ (R):

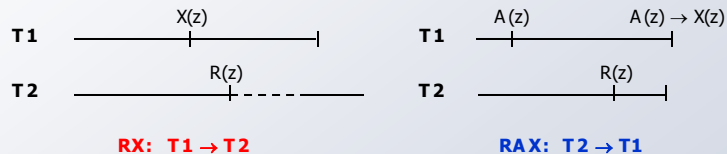
LRQ (A):

8-32

RAX (3)

Eigenschaften von RAX

- Paralleles Lesen der gültigen Version wird zugelassen
- Schreiben wird nach wie vor serialisiert (A-Sperre)
- Bei EOT Konversion der A- nach X-Sperren, ggf. auf Freigabe von Lesesperren warten (Deadlock-Gefahr)
- Höhere Parallelität als beim RX-Verfahren, jedoch i. Allg. **andere Serialisierbarkeitsreihenfolge**



Nachteile

- Neue Version wird für neu ankommende Leser erst verfügbar, wenn alte Version aufgegeben werden kann
 - Starke Behinderungen von Update-TAs durch **lange Leser** möglich
- ⇒ Bei TA-Mix von **langen Lesern** und **kurzen Schreibern** auf gemeinsamen Objekten bringt RAX keinen großen Vorteil

8-33

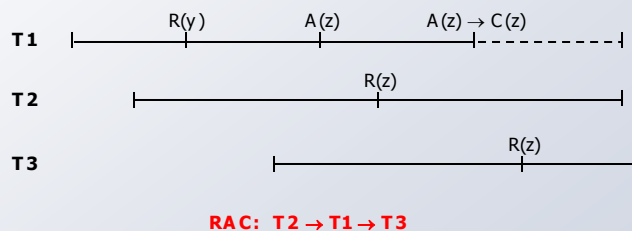
Sperrverfahren mit Versionen (RAC)

Kompatibilitätsmatrix

	R	A	C
R	+	+	⊕
A	+	-	-
C	⊕	-	-

C = Commit

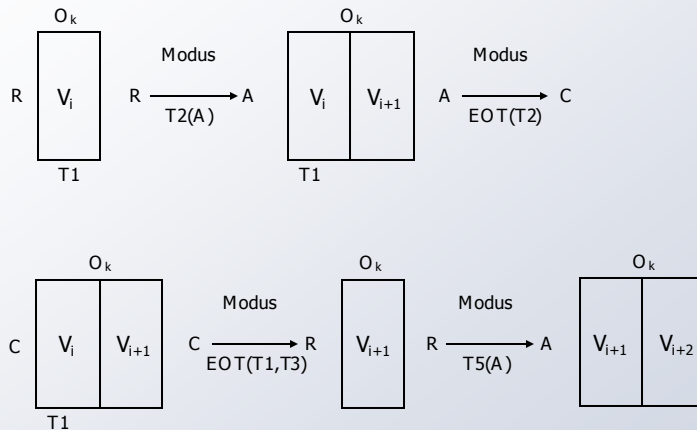
Ablaufbeispiel



8-34

RAC (2)

Änderungen erfolgen ebenfalls in temporärer Objektkopie



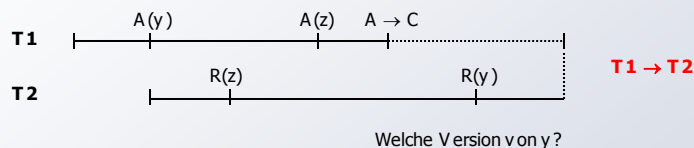
LRQ(A):

8-35

RAC (3)

Eigenschaften von RAC

- Änderungen werden nach wie vor serialisiert (A-Sperre erforderlich)
- Bei EOT Konversion von $A \rightarrow C$ -Sperre
- C-Sperre zeigt Existenz **zweier gültiger Objektversionen** an
- Maximal 2 Versionen, da C-Sperren mit sich selbst und mit A-Sperren unverträglich sind



⇒ **Kein Warten auf Freigabe von Lesesperren auf alter Version**
(R- und C-Modus sind verträglich)

Nachteile

- RAC ist nicht chronologieerhaltend
- Verwaltung komplexer Abhängigkeiten (z. B. über Abhängigkeitsgraphen)
⇒ **komplexere Sperrverwaltung**
- Leseanforderungen bewirken nie Blockierung/Rücksetzung, jedoch:
Auswahl der „richtigen“ Version erforderlich
- Änderungs-TAs, die auf C-Sperre laufen, müssen warten, **bis alle Leser der alten Version beendet sind** - weil nur 2 Versionen

⇒ **ABHILFE:** allgemeines Mehrversionen-V erfahren

8-36

Mehrversionen-Verfahren

Änderungs-TAs erzeugen neue Objektversionen

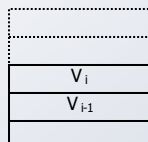
- Es kann immer nur eine neue Version pro Objekt erzeugt werden
- Sie wird bei EOT der TA freigegeben

Lese-TAs sehen den bei ihrem BOT gültigen DB-Zustand

- Sie greifen immer auf die jüngsten Objektversionen zu, die bei ihrem BOT freigegeben waren
⇒ Eine Objektversion, auf die ein Leser noch zugreifen könnte, darf nicht gelöscht werden
- Sie setzen und beachten keine Sperren
- Es gibt keine Blockierungen und Rücksetzungen für Lese-TAs, dafür ggf. Zugriff auf veraltete Objektversionen
- Wie erkennt das DBS Leser?

Beispiel für Objekt O_k

Zeitliche Reihenfolge der Zugriffe auf O_k



$T_j(\text{BOT})$	⇒	V_i (aktuelle Version)
$T_m(X)$	⇒	Erzeugen V_{i+1}
$T_n(X)$	⇒	Verzögern bis $T_m(\text{EOT})$
$T_m(\text{EOT})$	⇒	Freigeben V_{i+1}
$T_n(X)$	⇒	Erzeugen V_{i+2}
$T_j(\text{Ref})$	⇒	V_i
$T_n(\text{EOT})$	⇒	Freigeben V_{i+2}

8-37

Mehrversionen-Verfahren (2)

Kompatibilitätsmatrix für Objekt O

	NL	R	X
L	+	+	-
R	+	+	-
X	+	-	-

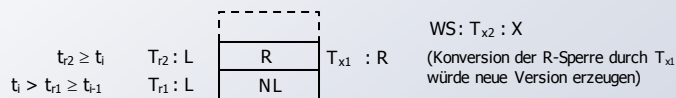
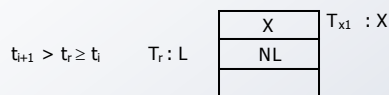
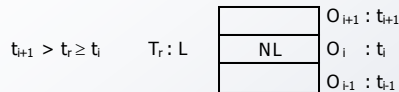
Auswahl der Objektversion

- Der Zugriff im Modus L bezieht sich auf die bei $\text{BOT}(T_r)$ freigegebene Version:
 - T_r : Lese-TA (Read-only-Operationen)
 - T_x : Schreib-TA (Read/Write-Operationen)
 - Commit-Zeitpunkte der Versionen: ... $O_i = t_i$, $O_{i+1} = t_{i+1}$...
 - $\text{BOT}(T_r) = t_r$
Wenn $t_{i+1} > t_r \geq t_i$, dann bezieht sich der L-Zugriff auf O_i

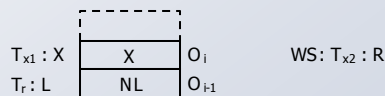
8-38

Mehrversionen-Verfahren (3)

■ Sperrsituationen



Zugriff auf O in der Reihenfolge $T_{x1}, T_{x2}, T_r (t_r \geq t_{i-1})$

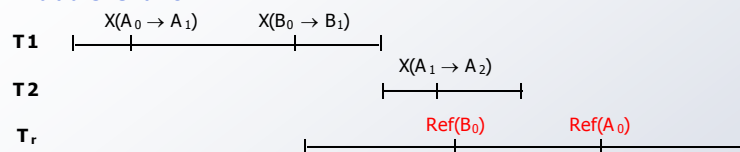


⇒ Mit R wird immer die jüngste freigegebene Version gelesen; solange $\text{Lock}(O, R)$ gilt, kann keine neue Version erzeugt werden

8-39

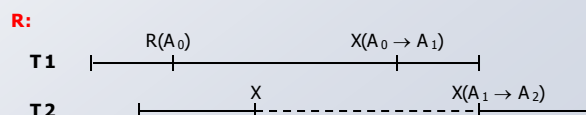
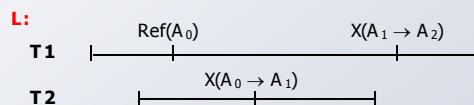
Mehrversionen-Verfahren (4)

■ Ablaufszenario



■ Konsequenz

- Lese-TAs werden bei der Synchronisation nicht mehr berücksichtigt
 - Schreib-TAs werden mit ihren Lese- und Schreib-Operationen untereinander über ein allgemeines Verfahren (Sperren, OCC, ...) synchronisiert
- ⇒ deutlich weniger Synchronisationskonflikte
- **Hypothetische Annahme:** Ein Schreiber greift bei seinen Lese-Operationen wie ein Leser zu (ohne Synchronisation). Ist der Schreiber serialisierbar?



8-40

Mehrversionen-Verfahren (5)

Zusätzlicher Speicher- und Wartungsaufwand

- Versionspoolverwaltung, *Garbage Collection*
- Auffinden von Versionen



- Speicherplatzoptimierung: Versionen auf Satzebene, Einsatz von Komprimierungstechniken
- Was passiert, wenn Implementierung auf n Versionen begrenzt ist?

8-41

Snapshot Isolation (SI)

Regeln

- 1) Wenn eine TA T ein Datenobjekt x liest, dann liefert das System immer den **jüngsten transaktionskonsistenten Wert** von x zum Zeitpunkt von $\text{BOT}(T)$. Dieser Wert von x stammt von der jüngsten erfolgreich beendeten TA unter allen TA, die x geschrieben haben und beendet waren, **bevor T gestartet wurde**. Wenn T selbst den Wert von x verändert, sieht es diesen geänderten Wert.
- 2) Wenn zwei TA T_i und T_j **zeitlich überlappen** und **dasselbe Datenobjekt schreiben**, verhindert das System, dass beide erfolgreich Commit ausführen können. In einem solchen Fall gilt „**First Committer Wins**“.

Notation

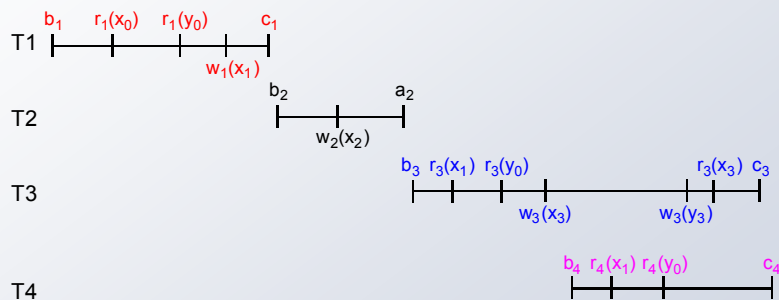
- r_i ist eine Leseoperation innerhalb von T_i
- Wenn T_j das Datenobjekt x schreibt, wird das durch $w_j(x_i)$ ausgedrückt
- Achtung: Der Index bei x_i zeigt nicht die Versionsnummer an. In einem Schedule bedeutet $r_5(x_3)$, dass TA T_5 den Wert von x liest, den T_3 geschrieben hat

8-42

Snapshot Isolation (2)

Allgemeine Wirkungsweise der SI (Synonym: Row Versioning)

S1: b_1 $r_1(x_0)$ $r_1(y_0)$ $w_1(x_1)$ c_1 b_2 $w_2(x_2)$ a_2 b_3 $r_3(x_1)$ $r_3(y_0)$ $w_3(x_3)$
 b_4 $r_4(x_1)$ $r_4(y_0)$ $w_3(y_3)$ $r_3(x_3)$ c_3 c_4



Besonderheiten

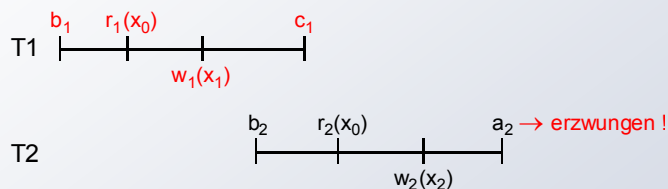
- Repeatable Read v von T3 (Regel 1)
- T4 liest x_1 , obwohl x_3 schon existiert (kein Warten wie bei Sperren)

8-43

Snapshot Isolation (3)

First Committer Wins!

S2: b_1 $r_1(x_0)$ $w_1(x_1)$ b_2 c_1 $r_2(x_0)$ $w_2(x_2)$ a_2



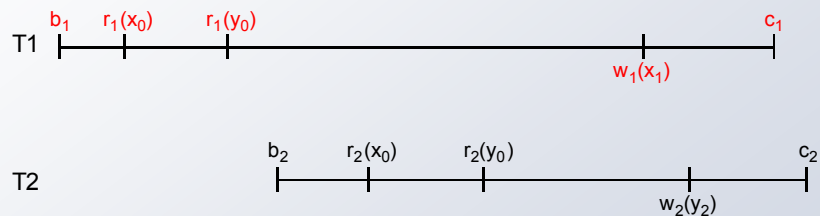
→ Commit v on T2 würde ein Lost Update v von T1 implizieren!

8-44

Snapshot Isolation (4)

- **Anomalie „Write Skew“:**
SI garantiert nicht Serialisierbarkeit!

S3: $b_1 \ r_1(x_0) \ r_1(y_0) \ b_2 \ r_2(x_0) \ r_2(y_0) \ w_1(x_1) \ w_2(y_2) \ c_1 \ c_2$



- **Annahme**

- x und y seien die Kontostände zweier Konten eines Kunden. Eine Geschäftsregel verlangt, dass die Gesamtsumme solcher zusammengehöriger Konten immer positiv ist.
- Wenn eine TA erkennt, dass die Gesamtsumme negativ ist, führt sie Abort aus
- Beispiel: $x_0 = 100$; $y_0 = 200$; $T1 \rightarrow x - 150$; $T2 \rightarrow y - 175$;

8-45

Snapshot Isolation (5)

- **Einsatz der SI in kommerziellen DBMS**

- "SET TRANSACTION ISOLATION LEVEL SNAPSHOT" in Microsoft SQL Server 2005
- Ähnliche Einsatzmöglichkeit in Interbase und Oracle Berkeley DB
- Andere Plattformen wie PostgreSQL (ab Version 7) und Oracle wenden Snapshot Isolation an, wenn der Benutzer "SET ISOLATION LEVEL SERIALIZABLE" spezifiziert
- Achtung: SI erlaubt nicht-serialisierbare TA-Ausführungen!

8-46

Prädikatssperren³

■ Logische Sperren oder Prädikatssperren

- Minimaler Sperrbereich durch geeignete Wahl des Prädikats
- **Verhütung des Phantomproblems**
- Eleganz

■ Form

LOCK (R, P, a)

R Tabellenname
P Prädikat
a ∈ {read, write}

UNLOCK(R, P)

- Lock (R, P, write) sperrt alle möglichen Sätze von R exklusiv, die Prädikat P erfüllen

■ Beispiel

T1:	LOCK(R1, P1, read)	T2:	...
	LOCK(R2, P2, write)		LOCK(R2, P3, write)
	LOCK(R1, P5, write)		LOCK(R1, P4, read)
	...		...

■ Wie kann Konflikt zwischen zwei Prädikaten festgestellt werden?

- Im allgemeinen Fall rekursiv unentscheidbar, selbst mit eingeschränkten arithmetischen Operatoren
- Entscheidbare Klasse von Prädikaten: einfache Prädikate
⇒ (A Θ Wert) {^, v} (. . .

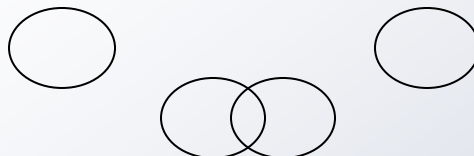
3. Eswaran, K.P. et al.: The Notions of Consistency and Predicate Locks in a Database System, in: Communications of the ACM 19:11, 1976, 624-633.

Prädikatssperren (2)

■ Entscheidungsprozedur

LOCK (R, P, a)

LOCK (R', P', a')



1. Wenn $R \neq R'$, kein Konflikt
2. Wenn $a = \text{read}$ und $a' = \text{read}$, kein Konflikt
3. Wenn $P(t) \wedge P'(t) = \text{TRUE}$ für irgendein t , dann besteht ein Konflikt

T1:	T2:
LOCK (Pers, Alter > 50, read)	LOCK (Pers, Pnr = 4711, write)

⇒ **Entscheidung:**

■ Nachteile

- Erfüllbarkeitstest: A aufwändige Entscheidungsprozedur mit vielen Prädikaten ($N > 100$) (wird in innerer Schleife des Lock-Mgr häufig aufgerufen)
- **Pessimistische Entscheidungen** ⇒ **Einschränkung der Parallelität** (es wird auf Erfüllbarkeit getestet!)
- Einsatz nur bei deskriptiven Sprachen!
- Sonderfall: $P = \text{TRUE}$ entspricht einer Tabellensperre
⇒ **große Sperrgranulate, geringe Parallelität**

Prädikatssperren (3)

■ Effizientere Implementierung: Präzisionssperren⁴

- sperren nur die gelesenen Daten durch Prädikate
- setzen für aktualisierte Sätze Schreibsperren
- ⇒ Es ist kein Disjunktheitstest für Prädikate mehr erforderlich, sondern es ist lediglich zu überprüfen, ob der Satz ein Prädikat erfüllt

■ Datenstrukturen

- **Prädikatsliste** Lesesperren laufender TAs werden durch Prädikate beschrieben

(Pers: Alter > 50 and Beruf = 'Prog.')
 (Pers: Pname = 'Meier' and Gehalt > 50000)
 (Abt: Anr=K55)
 . . .

- **Update-Liste** enthält geänderte Sätze laufender TAs

(Pers: 4711, 'Müller', 30, 'Prog.', 70000)
 (Abt: K51, 'DBS', . . .)
 . . .

■ Leseanforderung (Prädikat P):

- für jeden Satz der Update-Liste ist zu prüfen, ob er P erfüllt
- wenn ja ⇒ **Sperrkonflikt**

■ Schreibenanforderung (Satz T):

- für jedes Prädikat P der Prädikatsliste ist zu prüfen, ob T es erfüllt
- wenn T keines erfüllt ⇒ **Schreibsperre wird gewährt**

4. J.R. Jordan, J. Banerjee, R.B. Batmat: Precision Locks, in: Proc. ACM SIGMOD, 1981, 143-147

Synchronisation auf Objekten

■ Erhöhung der Parallelität durch Einführung kommutativer („semantischer“) DB-Operationen als Einheit für die Synchronisation

- Synchronisation erfolgt auf abstrakterer Ebene (**Objektebene**)
- Realisierung muss auf niedrigerer Ebene Korrektheit gewährleisten (z. B. durch Escrow-Verfahren)

■ Beispiel

Zwei Kontobuchungen auf Konten K1 und K2 durch die TAs T1 und T2, die **kommutative Operationen** „erhöhe um x“ und „vermindere um x“ verwenden, könnten z.B. in folgenden Reihenfolgen ausgeführt werden:

Schedule S1

1. erhöhe (T1,K1,x1)
2. vermindere (T1,K2,x1)
3. erhöhe (T2,K1,x2)
4. vermindere (T2,K2,x2)

Schedule S2

1. erhöhe (T1,K1,x1)
2. erhöhe (T2,K1,x2)
3. vermindere (T2,K2,x2)
4. vermindere (T1,K2,x1)

Synchronisation auf Objekten (2)

Synchronisation

Sperrverfahren

Hierarchische Verfahren

Konsistenz-ebenen

Optimierungen

Bewertung/Leistungsanalyse



⇒ Schedule S1 ist auch mit rein syntaktischen V erfahren (r/w) erzeugbar

$\underbrace{\langle r_1[K_1] \ w_1[K_1] \rangle}_{\text{erhöhe}(T_1, K_1, x_1)}$	$\underbrace{\langle r_1[K_2] \ w_1[K_2] \rangle}_{\text{vermindere}(T_1, K_2, x_1)}$	$\underbrace{\langle r_2[K_1] \ w_2[K_1] \rangle}_{\text{erhöhe}(T_2, K_1, x_2)}$	$\underbrace{\langle r_2[K_2] \ w_2[K_2] \rangle}_{\text{vermindere}(T_2, K_2, x_2)}.$
---	---	---	--

⇒ Schedule S2 ist hingegen mit rein syntaktischen V erfahren nicht erzeugbar

$\underbrace{\langle r_1[K_1] \ w_1[K_1] \rangle}_{\text{erhöhe}(T_1, K_1, x_1)}$	$\underbrace{\langle r_2[K_1] \ w_2[K_1] \rangle}_{\text{erhöhe}(T_2, K_1, x_2)}$	$\underbrace{\langle r_2[K_2] \ w_2[K_2] \rangle}_{\text{vermindere}(T_2, K_2, x_2)}$	$\underbrace{\langle r_1[K_2] \ w_1[K_2] \rangle}_{\text{vermindere}(T_1, K_2, x_1)}.$
---	---	---	--

8-51

Synchronisation von High-Traffic-Objekten

Synchronisation

Sperrverfahren

Hierarchische Verfahren

Konsistenz-ebenen

Optimierungen

Bewertung/Leistungsanalyse



■ High-Traffic-Objekte

- auch *Hot Spots* genannt: Datenelemente, auf die viele TAs ändernd zugreifen müssen
 - können leicht einen **Systemengpass** bilden
 - meist numerische Felder mit aggregierten Informationen z. B.
 - Anzahl freier Plätze
 - Summe aller Kontostände
 - aber auch DBS-interne Strukturen und Systemressourcen
 - Wurzeln von Bäumen (erfordern spezielle Protokolle)
 - **Log-Datei, Sperrtabelle, DB-Puffer** usw.
- ⇒ bei ungeschicktem TA-Scheduling (FIFO) besteht Gefahr der Ausbildung langer TA-Warteschlangen (**Konv oip hänomen**)

■ Einfachste Lösung der Synchronisationsprobleme

- Vermeidung solcher Datenelemente beim DB-Entwurf
- Einsatz von speziellen Sperren und angepassten Scheduling-Verfahren bei Systemressourcen

8-52

Synchronisation von High-Traffic-Objekten (2)

■ Was lässt sich auf SQL-Ebene tun?

- Modifikation eines Feldes FreiePlätze


```
exec sql  select FreiePlätze
          into :anzahl
          from Flüge
          where FlugNr = 'LH127';

          if anzahl ≥ 2 then anzahl := anzahl-2;

exec sql  update Flüge
          set FreiePlätze = :anzahl
          where FlugNr = 'LH127';
```

⇒ **Deadlock-Gefahr wegen Sperrkonversion. Erhöhung dieser Gefahr bei Hot-Spot-Verhalten**

- Verbesserung


```
exec sql  update Flüge
          set FreiePlätze = FreiePlätze-2
          where FlugNr = 'LH127';
```

aber: Prüfung der Integritätsbedingung ($\text{FreiePlätze} \geq 0$) durch das DBMS

⇒ **keine Deadlock-Gefahr, jedoch möglicherweise langes Warten auf Commit der Vorgänger-TAs.**

8-53

Synchronisation von High-Traffic-Objekten (3)

■ Spezielle Behandlung von Hot Spots

- erfordert Systemmodifikation
- **Aktion auf Hot Spot wird in zwei Teile zerlegt**
 - Testen eines Prädikats
 - Durchführung der Transformation
- Beispiel


```
exec sql  update hotspot Flüge
          set FreiePlätze = FreiePlätze-2
          where FlugNr = 'LH127'
          and FreiePlätze ≥ 2;
```

■ Ablauf des Attributzugriffs

1. Beim Zugriff erfolgt der **Test des Prädikats** unter einer kurzen Lesesperre
2. Falls der Test zu 'false' evaluiert wird, wird abgebrochen (Fehlerbehandlung der Anwendung)
3. Sonst wird ein **Redo-Log-Satz** mit Prädikat und Modifikationsoperation angelegt
4. **Beim Erreichen von Commit** werden die Transformationen von Hot-Spot-Feldern in zwei Phasen durchgeführt:

Phase 1: Alle Redo-Log-Sätze der TA, die Commit ausführt, werden abgearbeitet. Für Feldzugriffe, die keine Transformation erfordern, werden Lesesperren angefordert und für alle anderen Feldzugriffe Schreibsperren. Danach werden alle Prädikate noch einmal evaluiert. Falls mindestens ein Prädikat zu 'false' evaluiert, wird die TA zurückgesetzt. Sonst Eintritt in Phase 2.

Phase 2: Alle Transformationen werden angewendet und die Sperren werden freigegeben.

8-54

Synchronisation von High-Traffic-Objekten (4)

- **Beispiel:** 3 TAs werden gleichzeitig aufgerufen, aber sequentiell abgewickelt

T1: FP > 5 ?
FP := FP-5
Commit

T2: FP > 2 ?
FP := FP-2
Commit

T3: FP > 8 ?
FP := FP-8
Commit

- **Beispiel:** Spezielle Behandlung von *Hot Spots*

T1	T2	T3	FP in DB
FP > 5			12
	FP > 2 ? Commit FP > 2 FP := FP-2		
		FP > 8 ?	10
Commit FP > 5 ? FP := FP-5		Commit FP > 8 ?	5

8-55

Escrow-Ansatz⁵

- **High-Traffic-Objekte**

- Deklaration als Escrow-Felder
 - Benutzung spezieller Operationen
 - Anforderung einer bestimmten Wertmenge
IF **ESCROW** (field=F1, quantity=C1, test=(condition))
THEN 'continue with normal processing'
ELSE 'perform exception handling'
 - Benutzung der reservierten Wertmengen
USE (field=F1, quantity=C2)
 - Optionale Spezifikation eines Bereichstests bei Escrow-Anforderung
 - Wenn Anforderung erfolgreich ist, kann Prädikat nicht mehr nachträglich invalidiert werden
- ⇒ keine spätere Validierung/Zurücksetzung

- **Aktueller Wert eines Escrow-Feldes**

- ist unbekannt, wenn laufende TAs Reservierungen angemeldet haben
- ⇒ Führen eines Wertintervalls, das alle möglichen Werte nach Abschluss der laufenden TA umfasst
- für Wert Q_k des Escrow-Feldes k gilt:
$$LO_k \leq INF_k \leq Q_k \leq SUP_k \leq HI_k$$
- Anpassung von INF , Q , SUP bei Anforderung, Commit und Abort einer TA

Escrow-Ansatz (2)

■ Beispiel

- Zugriffe auf Feld mit LO=0, HI=100 (Anzahl freier Plätze)

Anforderungen/Rückgaben				Wertintervall		
T1	2	T3	T4	INF	Q	SUP
				15	15	15
-5				10	10	15
	-8			2	2	15
		+4		2	6	19
			-3			
commit				2	6	14
		commit		6	6	14
	abort			14	14	14

■ Eigenschaften

- Durchführung von Bereichstests bezüglich des Wertintervalls
- Minimal-/Maximalwerte (LO, HI) dürfen **nicht überschritten** werden
- hohe Parallelität ändernder Zugriffe möglich

■ Nachteile

- spezielle Programmierschnittstelle
- tatsächlicher Wert ggf. nicht abrufbar

8-57

Klassifikation von Synchronisationsverfahren

■ Gemeinsame Ziele

- Erhöhung der Parallelität
- Reduktion von Behinderungen/Blockierungen
- Einfache Verwaltung

■ Erhöhung der Parallelität durch Objektreplikation

- **Kopien:** temporär, privat, nicht freigegeben
- **Versionen:** permanent, mehrbenutzbar, freigegeben

#Versionen #Kopien	1	2	N	∞
0				
1				
P				

■ Beobachtung

- Einsatz in existierenden DBS: vor allem **hierarchische Sperrverfahren** und Varianten, aber auch **Mehrversionen-V erfahren**
- Es existieren eine Fülle von allgemeingültigen und spezialisierten Synchronisationsverfahren (zumindest in der Literatur)
- Es kommen (ständig) Verfahren durch **Variation der Grundprinzipien** dazu!

8-58

Implementierungsaspekte - Datenstrukturen

■ Probleme bei der Implementierung von Sperren

- Kleine Sperreinheiten (wünschenswert) erfordern hohen Aufwand
- Sperranforderung und -freigabe sollten sehr schnell erfolgen, da sie sehr häufig benötigt werden ⇒ **Sperrtabelle ist High-Traffic-Objekt!**
- Explizite, satzweise Sperren führen u. U. zu umfangreichen Sperrtabellen und großem Zusatzaufwand

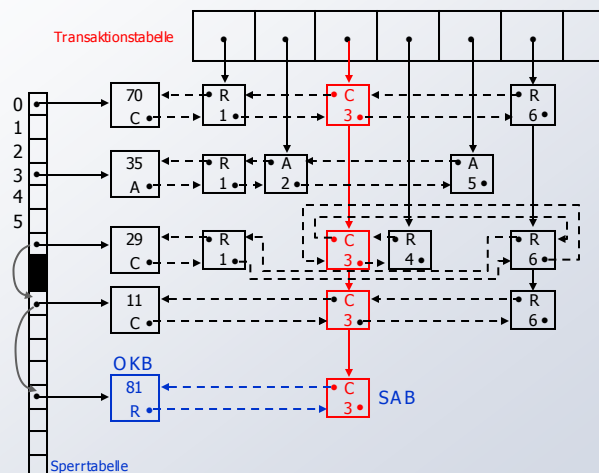
■ Beispiel: Sperrtabelle / TA-Tabelle für RAC-Verfahren

- Hash-Tabelle erlaubt schnellen Zugriff auf Objektkontrollblöcke (OKB)
- Matrixorganisation der Sperrtabelle mit Sperranforderungsblöcken (SAB)
- Spezielles Sperrverfahren: Kurzzeitsperren für Zugriffe auf Sperrtabelle (Semaphore pro Hash-Klasse reduziert Konflikt-/Konvoi-Gefahr)

8-59

Implementierungsaspekte - Datenstrukturen

■ Beispiel: Sperrtabelle / TA-Tabelle für RAC-Verfahren



8-60

Leistungsanalyse - Simulationsverfahren

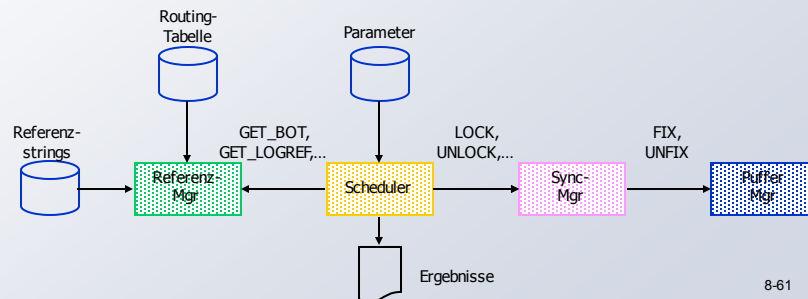
Analyse von Synchronisationsverfahren

- **pessimistisch:** RX, RX2
- **optimistisch:** BOCC, FOCC-K (Kill), FOCC-H (Hybrid)
- **Versionen:** RAC, Mehrversionen-Verfahren (MVC)

Nachbildung der DB-Last

- Aufzeichnung der Seitenreferenzen realer Anwendungen im DBS
- Nutzung verschiedenartiger TA-Mixe in Form von Referenzstrings
- Simulation des DB-Puffers und der benötigten E/A-Zeiten
- Ermittlung der Durchlaufzeiten unter den verschiedenen Synchronisationsverfahren und den eingestellten Sollparallelitäten

Simulationsverfahren



8-61

Leistungsanalyse und Bewertung von Synchronisationsverfahren

Wie bewertet man Parallelität?

- hoher Parallelitätsgrad – viele Rücksetzungen und Wiederholungen
- moderate Parallelität, dafür geringerer Zusatzaufwand

Durchsatztest

- alle Transaktionen inkl. (mehrfache) Wiederholungen sind bearbeitet (Ermittlung der Durchlaufzeit)
- einstellbarer Grad der maximalen Parallelität

Messung der effektiven Parallelität

- n = nominale Parallelität (MPL)
- n' = durchschnittliche Anzahl aktiver Transaktionen (berücksichtigt Wartesituationen)
- q = tatsächliche Arbeit (Referenzen) / minimale Arbeit (berücksichtigt Rücksetzungen und Wiederholungen)
- **effektive Parallelität**
 $n^* = n'/q$

Zählung der Deadlocks

8-62

Synchronisationsverfahren - Vergleich

Synchronisation

Sperrverfahren

Hierarchische Verfahren

Konsistenz-ebenen

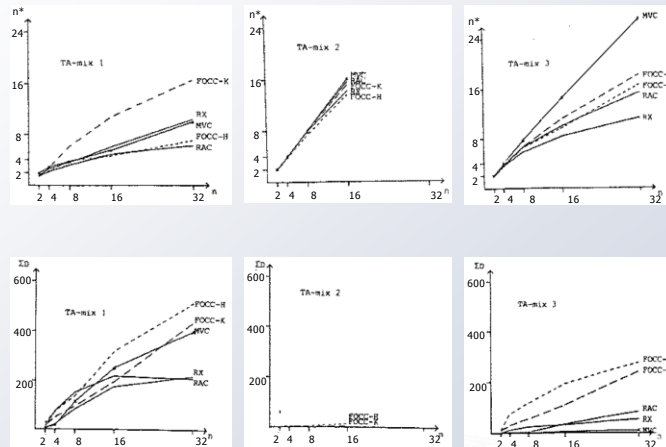
Optimierungen

Bewertung/Leistungsanalyse



Effektive Parallelität

Summe der Deadlocks



8-63

Synchronisationsverfahren - Vergleich

Synchronisation

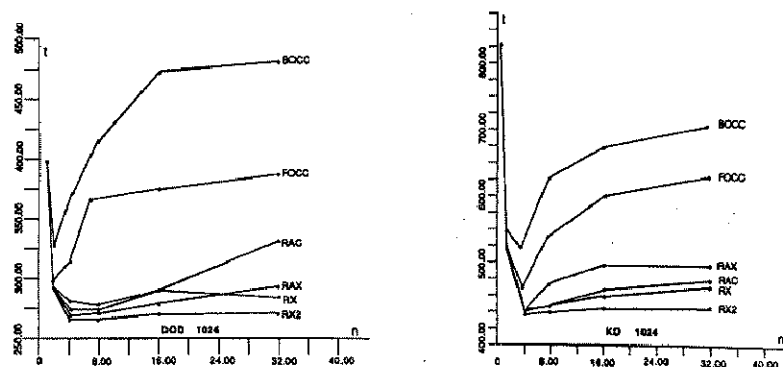
Sperrverfahren

Hierarchische Verfahren

Konsistenz-ebenen

Optimierungen

Bewertung/Leistungsanalyse



■ Schlussfolgerungen

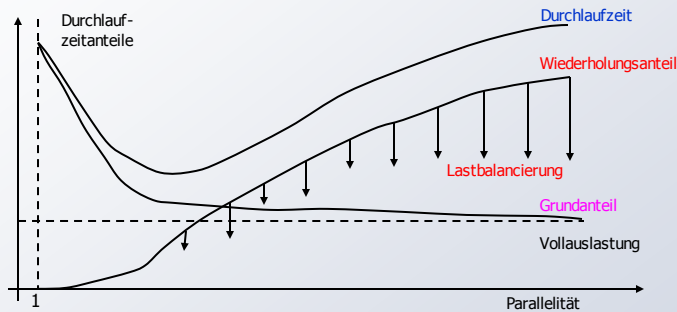
- Sehr geringe Parallelität ⇒ keine effektive Nutzung der Ressourcen
- Geringe Parallelität ⇒ bester Durchsatz, nicht notwendigerweise kürzeste Antwortzeiten
- Pessimistische Methoden gewinnen: Blockierung vermeidet häufig Deadlocks
- Optimistische Methoden geraten leicht in ein Thrashing-Verhalten
- RX2 reduziert effektiv den Wettbewerb um gemeinsam genutzte Daten

8-64

Synchronisation und Lastkontrolle

■ Charakteristische Wannenförmigkeit (idealisiert)

- Sie ergibt sich bei vielen Referenzstrings und Synchronisationsverfahren



- Sie wird von **zwei gegenläufigen Faktoren** bestimmt
 - Grundanteil der Durchlaufzeit beschreibt die Zeit, die zur Verarbeitung der durch den Referenzstring vorgegebenen Last bei fehlender Synchronisation nötig wäre
 - Wiederholungsanteil umfasst die Belegung des Prozessors zur nochmaligen Ausführung zurückgesetzter TAs
- ⇒ Rolle der Lastkontrolle und Lastbalancierung!

8-65

Synchronisation und Lastkontrolle (2)

■ Empirische Bestätigung

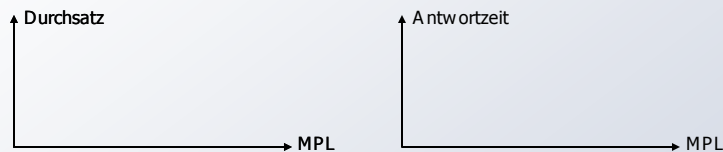
- Theoretische Untergrenze des Grundanteils wird bei Vollausslastung des Prozessors erreicht
- Häufigkeit von Leerphasen des Prozessors

DOD 1024							
PAR	BOCCT	FOCC	BOCC	RX2	RAX	RAC	RX
1	4864	4864	4864	4864	4864	4864	4864
2	1623	1734	1584	1719	1725	1717	1788
4	59	105	50	149	206	94	256
8	108	37	17	17	26	31	53
16	36	23	8	27	5	40	90
32	70	40	5	28	8	70	46

8-66

Dynamische Lastkontrolle

- **Was nützt „blinde“ Durchsatzmaximierung?**
- **Parallelitätsgrad** (*multiprogramming level, MPL*)
 - Er hat wesentlichen Einfluss auf das Leistungsverhalten, bestimmt Umfang der Konflikte bzw. Rücksetzungen
 - Gefahr von *Thrashing* bei Überschreitung eines **kritischen MPL-Wertes**



- **Statische MPL-Einstellung unzureichend:** wechselnde Lastsituationen, mehrere Transaktionstypen
- **Idee:** Dynamische Einstellung des MPL zur Vermeidung von *Thrashing*
- **Ein möglicher Ansatz**
Nutzung einer **Konflikttrate bei Sperrverfahren**⁶:

$$\text{Konflikttrate} = \frac{\text{\# gehaltener Sperren}}{\text{\# Sperren nicht-blockierter Transaktionen}}$$
 - kritischer Wert:** ca. 1,3 (experimentell bestimmt)
 - Zulassung neuer TAs nur, wenn kritischer Wert noch nicht erreicht ist
 - Bei Überschreiten erfolgt Abbrechen von TAs

6. Weikum, G. et al.: The Comfort Automatic Tuning Project, in: Information Systems 19:5, 1994, 381-432

Performanz vs. Korrektheit

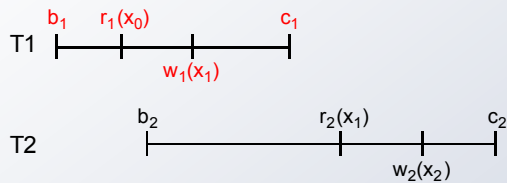
- **Trade-off**
 - Starke Isolation wie Serialisierbarkeit schützt die Datenintegrität
 - jedoch begrenzt den Durchsatz bei Konkurrenzsituationen (contention)
 - ➔ aber: **Serialisierbarkeit garantiert immer korrekte TA-Ergebnisse!**
- **Wunsch: Besserer Durchsatz auch bei „Contention“**
 - von Anfang an: Reduktion der Konsistenzebene (isolation level)
 - vor allem Read Committed (RC)
 - später: Mehrversionen-Verfahren (multiversion concurrency control algs.)
 - Snapshot Isolation (SI)
 - Multiversion Read Committed (RC_MV)*
- ➔ **Reduktion der Konsistenzebene erlaubt dem Anwendungsentwickler Domänenwissen zu nutzen; bei sorgfältigem Einsatz bleiben dabei TA-Ergebnisse / Daten auch korrekt**
- **Vergleich der Eigenschaften**
 - Eine TA T unter SI liest die Versionen der Daten, wie sie zu BOT(T) existierten.
 - RC_MV liest dagegen die jüngste freigegebene Version zum Zeitpunkt der Leseoperation (ähnlich wie das versionsfreie RC). Vergleiche mit SI!

* Oracle-Dokumentationen verwenden auch „Oracle Read Consistency“ für den RC_MV-Algorithmus

Performanz vs. Korrektheit (2)

■ Beispiel eines RC_MV-Schedules:

b_1 $r_1(x_0)$ $w_1(x_1)$ b_2 c_1 $r_2(x_1)$ $w_2(x_2)$ c_2

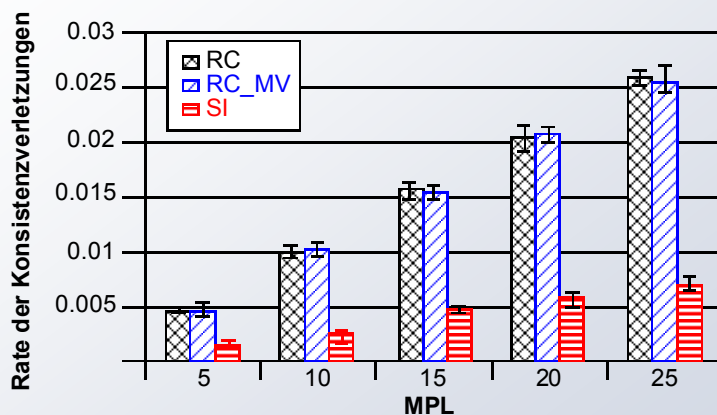


→ Im Allgemeinen ist aber SI deutlich strikter als RC und RC_MV, bei denen „Phantome“ und „inkonsistente Analysen“ auftreten können

8-69

Performanz vs. Korrektheit (3)

■ Beispiel: gemessene Rate der Konsistenzverletzungen vs MPL*



→ SI erzeugt hier wesentlich weniger Inkonsistenzen als RC und RC_MV

* Fekete, A., Goldrei, S.N., Asenjo, J.P.: Quantifying Isolation Anomalies. Proc. VLDB 2009, Lyon, France.

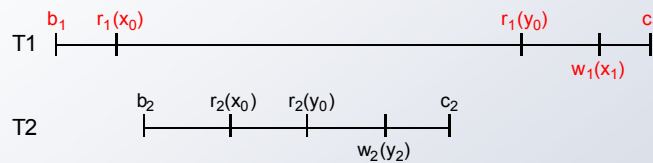
8-70

Performanz vs. Korrektheit (4)

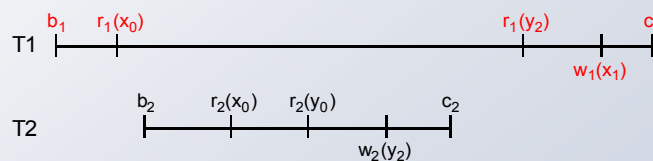
SI ist aber nicht durchgängig strikter als RC_MV und RC

Beispiel: b_1 $r_1(x)$ b_2 $r_2(x)$ $r_2(y)$ $w_2(y)$ c_2 $r_1(y)$ $w_1(x)$ c_1

- Ablauf unter SI



- Ablauf unter RC und RC_MV



8-71

Zusammenfassung

Realisierung der Synchronisation durch Sperrverfahren

- Prädikatssperren verkörpern eine elegante Idee, sind aber in praktischen Fällen nicht direkt einsetzbar, ggf. Nutzung in der Form von Präzisionssperren
- RAX und RAC begrenzen Anzahl der Versionen und reduzieren Blockierungsdauern nur für bestimmte Situationen
- **Mehrversionen-Verfahren** liefert hervorragende Werte bei der effektiven Parallelität und bei der Anzahl der Deadlocks, verlangt jedoch höheren Aufwand (Algorithmus, Speicherplatz)

⇒ Standard: multiple Sperrgranulate durch hierarchische Sperrverfahren

Einführung von Konsistenzebenen

- zwei (geringfügig) unterschiedliche Ansätze
 - basierend auf Sperrdauer für R und X
 - basierend auf zu tolerierende "Fehler"
- „Schwächere“ Synchronisation von Leseoperationen erlaubt höhere Parallelitätsgrade und Reduktion von Blockierungen, erfordert aber Programmierdisziplin!

⇒ Inkaufnahme von Anomalien reduziert die TA-Behinderungen

Generelle Optimierungen

- reduzierte Konsistenzebene und Mehrversionen-Ansatz
- aber: Trade-off von Performanz und Korrektheit beachten

'Harte' Synchronisationsprobleme

- Hot Spots / High-Traffic-Objekte
- lange (Änderungs-) TAs
- Wenn Vermeidungsstrategie nicht möglich ist, sind zumindest für Hochleistungssysteme Spezialprotokolle anzuwenden

Dynamische Lastkontrolle erforderlich

8-72